

# Python: module cdms.axis

## [\*cdms.axis\*](#)

[index](#)

CDMS [Axis](#) objects

### *Modules*

[MA](#)

[Numeric](#)

[PropertiedClasses](#)

[cdms.cdmsNode](#)

[cdms.cdmsobj](#)

[cdtime](#)

[copy](#)

[cdms.internattr](#)

[regrid](#)

[string](#)

[sys](#)

[types](#)

### *Classes*

[UserList.UserList](#)

[AliasList](#)

[cdms.cdmsobj.CdmsObj\(cdms.internattr.InternalAttributesClass\)](#)

[AbstractAxis](#)

[Axis](#)

[FileAxis](#)

[FileVirtualAxis](#)

[TransientAxis](#)

[TransientVirtualAxis](#)

class ***AbstractAxis***([cdms.cdmsobj.CdmsObj](#))

Method resolution order:

[AbstractAxis](#)

[cdms.cdmsobj.CdmsObj](#)

[cdms.internattr.InternalAttributesClass](#)

[PropertiedClasses.Properties.PropertiedClass](#)

---

Methods defined here:

[\\_\\_getitem\\_\\_\(self, key\)](#)

[\\_\\_getslice\\_\\_\(self, low, high\)](#)

[\\_\\_init\\_\\_\(self, parent, node\)](#)

[\\_\\_len\\_\\_\(self\)](#)

[\\_\\_repr\\_\\_ = \\_\\_str\\_\\_\(self\)](#)

```

__setitem__(self, index, value)
__setslice__(self, low, high, value)
__str__(self)

asComponentTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asDTGTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asRelativeTime(self)
    Array version of cdtime torel. Returns a list of component times.

assignValue(self, data)

clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient axis.
    If copyData is 1, make a separate copy of the data.

designateCircular(self, modulo, persistent=0)

designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0, modulo=360.0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.
    # If modulo is defined, set as circular

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar.
    # If persistent is true, write metadata to the container.

genGenericBounds(self, width=1.0)
    # Generate bounds from midpoints. width is the width of the zone.

getBounds(self)

getCalendar(self)
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,
    # or None. If the axis does not have a calendar attribute, return
    # calendar.

getData(self)

```

```

getExplicitBounds(self)
    # Return None if not explicitly defined

getModulo(self)

getModuloCycle(self)

getValue(self)
    # Return the entire array

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if
    applicable.

isCircular(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLinear(self)

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

isVirtual(self)
    Return true iff coordinate values are implicitly defined.

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='ccn', cycle=None)
    Map coordinate interval to index interval. interval has one of
    the following forms:
        (x, y)
        (x, y, indicator): indicator overrides keyword argument
        (x, y, indicator, cycle): indicator, cycle override keyword arguments
        None: indicates the full interval

```

where  $x$  and  $y$  are the endpoints in coordinate space. indicates two-character string, where the first character is 'c' if the axis is closed on the left, 'o' if open, and the second character has the same meaning for the right-hand point. Set cycle to a nonzero value to force wraparound.

Returns the corresponding index interval  $(i, j)$ , where  $i < j$ , indicating the half-open index interval  $[i, j)$ , or None if the intersection is empty.

For an axis which is circular (`self.topology == 'circular'`), the interval is interpreted as follows (where  $N = \text{len}(self)$ ):

- (1) if  $j \leq N$ , the interval does not wrap around the axis endpoint
- (2) if  $j > N$ , the interval wraps around, and is equivalent to the union of two consecutive intervals  $[i, N)$ ,  $[0, j-N)$

For example, if the vector is  $[0, 2, 4, \dots, 358]$  of length 180, and the coordinate interval is  $[-5, 5)$ , the return index interval is  $[178, 183)$ . This is equivalent to the two intervals  $[178, 180)$  and  $[0, 2)$ .

Note: if the interval is interior to the axis, but does not span the entire axis element, a singleton  $(i, i+1)$  indicating an adjacent index interval.

**`mapIntervalExt(self, interval, indicator='ccn', cycle=None, epsilon=None)`**

Like `mapInterval`, but returns  $(i, j, k)$  where  $k$  is stride, and  $(i, j)$  is not restricted to one cycle.

**`rank(self)`**

**`setBounds(self, bounds)`**

**`setCalendar(self, calendar, persistent=1)`**

# Set the calendar

**`subAxis = subaxis(self, i, j, k=1)`**

**`subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$ . The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

**`toRelativeTime(self, units, calendar=None)`**

Convert time axis values to another unit

**`typecode(self)`**

**`validateBounds(self, bounds)`**

# Check that a boundary array is valid, raise exception if not

---

Methods inherited from `cdms.cdmsobj.CdmsObj`:

**`dump(self, path=None, format=1)`**

```

dump(self, path=None, format=1)
    Dump an XML representation of this object to a file.
    'path' is the result file name, None for standard output.
    'format'==1 iff the file is formatted with newlines for readability.

matchPattern(self, pattern, attribute, tag)
    # Match a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

matchone(self, pattern, attrname)
    # Return true iff the attribute with name attrname is a string
    # attribute which matches the compiled regular expression pattern
    # if attrname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # search all string attributes. If tag is not None, it must match

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, attrname)
    Return true iff the attribute with name attrname is a string
    attribute which contains the compiled regular expression pattern
    if attrname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

---

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

```

```
get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
    nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
    nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

## class **AliasList**(UserList.UserList)

Methods defined here:

```
__init__(self, alist)

__setitem__(self, i, value)

append(self, value)

extend(self, values)
```

---

Methods inherited from UserList.UserList:

```
__add__(self, other)

__cmp__(self, other)

__contains__(self, item)

__delitem__(self, i)

__delslice__(self, i, j)

__eq__(self, other)

__ge__(self, other)

__getitem__(self, i)

__getslice__(self, i, j)

__gt__(self, other)
```

```
__iadd__(self, other)
__imul__(self, n)
__le__(self, other)
__len__(self)
__lt__(self, other)
__mul__(self, n)
__ne__(self, other)
__radd__(self, other)
__repr__(self)
__rmul__ = __mul__(self, n)
__setslice__(self, i, j, other)
count(self, item)
index(self, item, *args)
insert(self, i, item)
pop(self, i=-1)
remove(self, item)
reverse(self)
sort(self, *args, **kwds)
```

```
class Axis(AbstractAxis)
```

```
# One-dimensional coordinate axis in a dataset
```

Method resolution order:

```
Axis
AbstractAxis
cdms.cdmsobj.CdmsObj
cdms.internattr.InternalAttributesClass
PropertiedClasses.Properties.PropertiedClass
```

Methods defined here:

```
__getitem__(self, key)
```

```

        # Handle slices of the form x[i], x[i:j:k], and x[...]

getslice(self, low, high)
    # Handle slices of the form x[i:j]

init(self, parent, axisNode=None)

len(self)

getBounds(self)
    # Return the bounds array, or generate a default if autoBound

getCalendar(self)

getData(self)
    # Get axis data

getExplicitBounds(self)
    # Return the bounds array, or None

isLinear(self)
    # Return true iff the axis representation is linear

typecode(self)

```

---

Methods inherited from [AbstractAxis](#):

```

repr = str(self)

setitem(self, index, value)

setslice(self, low, high, value)

str(self)

asComponentTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asDTGTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asRelativeTime(self)
    Array version of cdtime torel. Returns a list of component times.

assignValue(self, data)

clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient axis.
    If copyData is 1, make a separate copy of the data.

designateCircular(self, modulo, persistent=0)

```

```

designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0, modulo=360.0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.
    # If modulo is defined, set as circular

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar
    # If persistent is true, write metadata to the container.

genGenericBounds(self, width=1.0)
    # Generate bounds from midpoints. width is the width of the zone

getModulo(self)

getModuloCycle(self)

getValue(self)
    # Return the entire array

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if applicable

isCircular(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude or latitude axis

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude or latitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)

```

```

# Return true iff the axis is a time axis

isVirtual(self)
    Return true iff coordinate values are implicitly defined.

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='ccn', cycle=None)
    Map coordinate interval to index interval. interval has one of
        (x,y)
        (x,y,indicator): indicator overrides keyword argument
        (x,y,indicator,cycle): indicator, cycle override keyword arguments
        None: indicates the full interval
    where x and y are the endpoints in coordinate space. indicator is a two-character string, where the first character is 'c' if the left endpoint is closed, 'o' if open, and the second character has the same meaning for the right-hand point. Set cycle to a nonzero value to force wraparound.

    Returns the corresponding index interval (i,j), where i<j, in the half-open index interval [i,j), or None if the intersection is empty.

    For an axis which is circular (self.topology == 'circular'), this is interpreted as follows (where N = len(self)):
        (1) if j<=N, the interval does not wrap around the axis endpoint
        (2) if j>N, the interval wraps around, and is equivalent to the union of two consecutive intervals [i,N), [0,j-N)
    For example, if the vector is [0,2,4,...,358] of length 180, and the coordinate interval is [-5,5), the return index interval is [178,183). This is equivalent to the two intervals [178,180).

    Note: if the interval is interior to the axis, but does not start at an axis element, a singleton (i,i+1) indicating an adjacent index interval is returned.

mapIntervalExt(self, interval, indicator='ccn', cycle=None, epsilon=None)
    Like mapInterval, but returns (i,j,k) where k is stride, and (i,j) is not restricted to one cycle.

rank(self)

setBounds(self, bounds)

setCalendar(self, calendar, persistent=1)
    # Set the calendar

subAxis = subaxis(self, i, j, k=1)
    Create a transient axis for the index slice [i:j:k]
    The stride k can be positive or negative. Wraparound is

```

supported for longitude dimensions or those with a modulus at

***subaxis*(self, i, j, k=1)**

Create a transient axis for the index slice [i:j:k]

The stride k can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus at

***toRelativeTime*(self, units, calendar=None)**

Convert time axis values to another unit

***validateBounds*(self, bounds)**

# Check that a boundary array is valid, raise exception if no

---

Methods inherited from cdms.cdmsobj.CdmsObj:

***dump*(self, path=None, format=1)**

dump(self, path=None, format=1)

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability

***matchPattern*(self, pattern, attribute, tag)**

# Match a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match tag

***matchone*(self, pattern, attrname)**

# Return true iff the attribute with name attrname is a string attribute which matches the compiled regular expression pattern. If attrname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

***searchPattern*(self, pattern, attribute, tag)**

# Search for a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match tag

***searchPredicate*(self, predicate, tag)**

# Apply a truth-valued predicate. Return a list containing a string if the predicate is true and either tag is None or matches tag. If the predicate returns false, return an empty list

***searchone*(self, pattern, attrname)**

Return true iff the attribute with name attrname is a string attribute which contains the compiled regular expression pattern. If attrname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

---

Methods inherited from cdms.internattr.InternalAttributesClass:

***is\_internal\_attribute*(self, name)**

is\_internal\_attribute(name) is true if name is internal.

```
replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
        Replace the external attributes with dictionary newAttributes
```

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

**\_\_delattr\_\_**(self, name)

**\_\_getattr\_\_**(self, name)

**\_\_setattr\_\_**(self, name, value)

**get\_property\_d**(self, name)

    Return the 'del' property handler for name that self uses.  
    Returns None if no handler.

**get\_property\_g**(self, name)

    Return the 'get' property handler for name that self uses.  
    Returns None if no handler.

**get\_property\_s**(self, name)

    Return the 'set' property handler for name that self uses.  
    Returns None if no handler.

**set\_property**(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)

    Set attribute handlers for name to methods actg, acts, actd  
    None means no change for that action.  
    nowrite = 1 prevents setting this attribute.  
        nowrite defaults to 0.  
    nodelete = 1 prevents deleting this attribute.  
        nodelete defaults to 1 unless actd given.  
    if nowrite and nodelete is None: nodelete = 1

class ***FileAxis***(AbstractAxis)

# One-dimensional coordinate axis in a CdmsFile.

Method resolution order:

[FileAxis](#)

[AbstractAxis](#)

[cdms.cdmsobj.CdmsObj](#)

[cdms.internattr.InternalAttributesClass](#)

[PropertiedClasses.Properties.PropertiedClass](#)

---

Methods defined here:

**\_\_delattr\_\_**(self, name)

# delattr deletes external global attributes in the file

```

__getitem__(self, key)
    # Read data
    # If the axis has a related Cdunif variable object, just read
    # otherwise, cache the Cdunif (read-only) data values in self
    # the axis is not extensible, so it is not necessary to reread

__getslice__(self, low, high)

__init__(self, parent, axisname, obj=None)

__len__(self)

__setattr__(self, name, value)
    # setattr writes external attributes to the file

__setitem__(self, index, value)

__setslice__(self, low, high, value)

getBounds(self)
    # Return the bounds array, or generate a default if autobounds is None

getCalendar(self)

getData(self)

getExplicitBounds(self)
    # Return the bounds array, or None

isLinear(self)

isVirtual(self)
    Return true iff coordinate values are implicitly defined.

setBounds(self, bounds, persistent=0, validate=0, index=None, boundsid=None)
    # Create and write boundary data variable. An exception is raised
    # if the bounds are already set. bounds is a Numeric array.
    # If persistent==1, write to file, else save in self._boundsArray
    # For a persistent axis, index=n writes the bounds starting at dimension n
    # index in the extended dimension (default is index=0).
    # If the bounds variable is new, use the name boundsid, or 'bounds'
    # if unspecified.

typecode(self)

```

---

Methods inherited from [AbstractAxis](#):

```

__repr__ = __str__(self)

__str__(self)

asComponentTime(self, calendar=None)

```

```

        Array version of cdtime tocomp. Returns a list of component times.

asDTGTime(self, calendar=None)
        Array version of cdtime tocomp. Returns a list of component times.

asRelativeTime(self)
        Array version of cdtime torel. Returns a list of component times.

assignValue(self, data)

clone(self, copyData=1)
        clone (self, copyData=1)
        Return a copy of self as a transient axis.
        If copyData is 1, make a separate copy of the data.

designateCircular(self, modulo, persistent=0)

designateLatitude(self, persistent=0)
        # Designate axis as a latitude axis.
        # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
        # Designate axis as a vertical level axis
        # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0, modulo=360.0)
        # Designate axis as a longitude axis
        # If persistent is true, write metadata to the container.
        # If modulo is defined, set as circular

designateTime(self, persistent=0, calendar=4369)
        # Designate axis as a time axis, and optionally set the calendar.
        # If persistent is true, write metadata to the container.

genGenericBounds(self, width=1.0)
        # Generate bounds from midpoints. width is the width of the zones.

getModulo(self)

getModuloCycle(self)

getValue(self)
        # Return the entire array

info(self, flag=None, device=None)
        Write info about axis; include dimension values and weights if applicable.

isCircular(self)
        # Return true iff the axis wraps around
        # An axis is defined as circular if:
        # (1) self.topology=='circular', or
        # (2) self.topology is undefined, and the axis is a longitude or latitude

```

```

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='ccn', cycle=None)
    Map coordinate interval to index interval. interval has one of
    three forms:
        (x,y)
        (x,y,indicator): indicator overrides keyword argument
        (x,y,indicator,cycle): indicator, cycle override keyword arguments
        None: indicates the full interval
    where x and y are the endpoints in coordinate space. indicator is a two-character string, where the first character is 'c' if the left endpoint is closed, 'o' if open, and the second character has the same meaning for the right-hand point. Set cycle to a nonzero value to force wraparound.

    Returns the corresponding index interval (i,j), where i<j, in the half-open index interval [i,j), or None if the intersection is empty.

    For an axis which is circular (self.topology == 'circular'), is interpreted as follows (where N = len(self)):
        (1) if j<=N, the interval does not wrap around the axis endpoint
        (2) if j>N, the interval wraps around, and is equivalent to the union of two consecutive intervals [i,N), [0,j-N)
    For example, if the vector is [0,2,4,...,358] of length 180, and the coordinate interval is [-5,5), the return index interval is [178,183). This is equivalent to the two intervals [178,180)

    Note: if the interval is interior to the axis, but does not start at an axis element, a singleton (i,i+1) indicating an adjacent index interval is returned.

mapIntervalExt(self, interval, indicator='ccn', cycle=None, epsilon=None)

```

Like `mapInterval`, but returns  $(i, j, k)$  where  $k$  is stride, and  $(i, j)$  is not restricted to one cycle.

**`rank(self)`**

**`setCalendar(self, calendar, persistent=1)`**

# Set the calendar

**`subAxis = subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus at

**`subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus at

**`toRelativeTime(self, units, calendar=None)`**

Convert time axis values to another unit

**`validateBounds(self, bounds)`**

# Check that a boundary array is valid, raise exception if no

---

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

**`dump(self, path=None, format=1)`**

**`dump(self, path=None, format=1)`**

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability

**`matchPattern(self, pattern, attribute, tag)`**

# Match a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match

**`matchone(self, pattern, attrname)`**

# Return true iff the attribute with name attrname is a string and matches pattern

# attribute which matches the compiled regular expression pattern

# if attrname is None and pattern matches at least one string

# attribute. Return false if the attribute is not found or is not a string

**`searchPattern(self, pattern, attribute, tag)`**

# Search for a pattern in a string-valued attribute. If attribute is None, search all string attributes. If tag is not None, it must match

**`searchPredicate(self, predicate, tag)`**

# Apply a truth-valued predicate. Return a list containing a

# if the predicate is true and either tag is None or matches

# If the predicate returns false, return an empty list

**`searchone(self, pattern, attrname)`**

Return true iff the attribute with name atname is a string attribute which contains the compiled regular expression pattern if atname is None and pattern matches at least one string attribute. Return false if the attribute is not found or is not a string.

---

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

***is\_internal\_attribute***(self, name)

`is internal attribute(name)` is true if name is internal.

***replace\_external\_attributes***(self, newAttributes)

`replace external attributes(newAttributes)`

Replace the external attributes with dictionary newAttributes

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

***\_\_getattr\_\_***(self, name)

***get\_property\_d***(self, name)

Return the 'del' property handler for name that self uses.  
Returns None if no handler.

***get\_property\_g***(self, name)

Return the 'get' property handler for name that self uses.  
Returns None if no handler.

***get\_property\_s***(self, name)

Return the 'set' property handler for name that self uses.  
Returns None if no handler.

***set\_property***(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)

Set attribute handlers for name to methods actg, acts, actd  
None means no change for that action.

nowrite = 1 prevents setting this attribute.

nowrite defaults to 0.

nodelete = 1 prevents deleting this attribute.

nodelete defaults to 1 unless actd given.

if nowrite and nodelete is None: nodelete = 1

---

class ***FileVirtualAxis***([FileAxis](#))

An axis with no explicit representation of data values in the file.  
It appears to be a float vector with values [0.0, 1.0, ..., float(  
This is especially useful for the bound axis used with boundary var.  
For a netCDF file the representation is a dimension with no associated  
coordinate variable.

---

Method resolution order:

[FileVirtualAxis](#)  
[FileAxis](#)  
[AbstractAxis](#)  
[cdms.cdmsobj.CdmsObj](#)  
[cdms.internattr.InternalAttributesClass](#)  
[PropertiedClasses.Properties.PropertiedClass](#)

---

Methods defined here:

**\_\_init\_\_**(self, parent, axisname, axislen)

**\_\_len\_\_**(self)

**getData**(self)

**isVirtual**(self)

Return true iff coordinate values are implicitly defined.

---

Methods inherited from [FileAxis](#):

**\_\_delattr\_\_**(self, name)

# delattr deletes external global attributes in the file

**\_\_getitem\_\_**(self, key)

# Read data

# If the axis has a related Cdunif variable object, just read  
# otherwise, cache the Cdunif (read-only) data values in self  
# the axis is not extensible, so it is not necessary to reread

**\_\_getslice\_\_**(self, low, high)

**\_\_setattr\_\_**(self, name, value)

# setattr writes external attributes to the file

**\_\_setitem\_\_**(self, index, value)

**\_\_setslice\_\_**(self, low, high, value)

**getBounds**(self)

# Return the bounds array, or generate a default if autobounds

**getCalendar**(self)

**getExplicitBounds**(self)

# Return the bounds array, or None

**isLinear**(self)

**setBounds**(self, bounds, persistent=0, validate=0, index=None, boundsid=None)

# Create and write boundary data variable. An exception is raised  
# if the bounds are already set. bounds is a Numeric array.

```

# If persistent==1, write to file, else save in self._bounds
# For a persistent axis, index=n writes the bounds starting at
# index in the extended dimension (default is index=0).
# If the bounds variable is new, use the name boundsid, or 'b'
# if unspecified.

typecode(self)



---


Methods inherited from AbstractAxis:

__repr__ = __str__(self)

__str__(self)

asComponentTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asDTGTime(self, calendar=None)
    Array version of cdtime tocomp. Returns a list of component times.

asRelativeTime(self)
    Array version of cdtime torel. Returns a list of component times.

assignValue(self, data)

clone(self, copyData=1)
    clone (self, copyData=1)
    Return a copy of self as a transient axis.
    If copyData is 1, make a separate copy of the data.

designateCircular(self, modulo, persistent=0)

designateLatitude(self, persistent=0)
    # Designate axis as a latitude axis.
    # If persistent is true, write metadata to the container.

designateLevel(self, persistent=0)
    # Designate axis as a vertical level axis
    # If persistent is true, write metadata to the container.

designateLongitude(self, persistent=0, modulo=360.0)
    # Designate axis as a longitude axis
    # If persistent is true, write metadata to the container.
    # If modulo is defined, set as circular

designateTime(self, persistent=0, calendar=4369)
    # Designate axis as a time axis, and optionally set the calendar.
    # If persistent is true, write metadata to the container.

genGenericBounds(self, width=1.0)
    # Generate bounds from midpoints. width is the width of the zone.

```

```

getModulo(self)

getModuloCycle(self)

getValue(self)
    # Return the entire array

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if
    available.

isCircular(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='ccn', cycle=None)
    Map coordinate interval to index interval. interval has one of
    three forms:
        (x,y)
        (x,y,indicator): indicator overrides keyword argument
        (x,y,indicator,cycle): indicator, cycle override keyword arguments
        None: indicates the full interval

    where x and y are the endpoints in coordinate space. indicator is a
    two-character string, where the first character is 'c' if the left-hand
    point is closed on the left, 'o' if open, and the second character has
    the same meaning for the right-hand point. Set cycle to a nonzero value
    to force wraparound.

    Returns the corresponding index interval (i,j), where i<j, in
    coordinate space.

```

the half-open index interval [i,j), or None if the intersection is empty.

For an axis which is circular (self.**topology** == 'circular'), is interpreted as follows (where N = len(self)):

- (1) if j<=N, the interval does not wrap around the axis endpoint i.
- (2) if j>N, the interval wraps around, and is equivalent to the two consecutive intervals [i,N), [0,j-N)

For example, if the vector is [0,2,4,...,358] of length 180, and the coordinate interval is [-5,5), the return index interval is [178,183). This is equivalent to the two intervals [178,180).

Note: if the interval is interior to the axis, but does not span the entire axis element, a singleton (i,i+1) indicating an adjacent index.

#### **mapIntervalExt**(self, interval, indicator='ccn', cycle=None, epsilon=None)

Like `mapInterval`, but returns (i,j,k) where k is stride, and (i,j) is not restricted to one cycle.

#### **rank**(self)

#### **setCalendar**(self, calendar, persistent=1)

# Set the calendar

#### **subAxis** = **subaxis**(self, i, j, k=1)

Create a transient axis for the index slice [i:j:k]

The stride k can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

#### **subaxis**(self, i, j, k=1)

Create a transient axis for the index slice [i:j:k]

The stride k can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

#### **toRelativeTime**(self, units, calendar=None)

Convert time axis values to another unit

#### **validateBounds**(self, bounds)

# Check that a boundary array is valid, raise exception if not.

---

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

#### **dump**(self, path=None, format=1)

[dump](#)(self, path=None, format=1)

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

#### **matchPattern**(self, pattern, attribute, tag)

# Match a pattern in a string-valued attribute. If attribute is None,

# search all string attributes. If tag is not None, it must match tag.

```

matchone(self, pattern, atname)
    # Return true iff the attribute with name atname is a string
    # attribute which matches the compiled regular expression pattern
    # if atname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not
    # a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute
    # is None, search all string attributes. If tag is not None, it must
    # match the tag of the attribute.

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # string if the predicate is true and either tag is None or matches
    # the tag of the attribute. If the predicate returns false, return an empty list.

searchone(self, pattern, atname)
    Return true iff the attribute with name atname is a string
    attribute which contains the compiled regular expression pattern
    if atname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not
    a string.

```

---

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__getattr__(self, name)
get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd.
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.

```

```
    nowrite defaults to 0.  
    nodelete = 1 prevents deleting this attribute.  
    nodelete defaults to 1 unless actd given.  
    if nowrite and nodelete is None: nodelete = 1
```

```
class TransientAxis(AbstractAxis)  
    # In-memory coordinate axis
```

Method resolution order:

```
TransientAxis  
AbstractAxis  
cdms.cdmsobj.CdmsObj  
cdms.internattr.InternalAttributesClass  
PropertiedClasses.Properties.PropertiedClass
```

Methods defined here:

```
__getitem__(self, key)  
__getslice__(self, low, high)  
__init__(self, data, bounds=None, id=None, attributes=None, copy=0)  
__len__(self)  
__setitem__(self, index, value)  
__setslice__(self, low, high, value)  
getBounds(self)  
getData(self)  
getExplicitBounds(self)  
isLinear(self)  
setBounds(self, bounds, persistent=0, validate=0, index=None, boundsid=None)  
    # Set bounds. The persistent argument is for compatibility with  
    # persistent versions, is ignored. Same for boundsid and index.  
    #  
    # mf 20010308 - add validate key word, by default do not validate  
typecode(self)
```

Data and other attributes defined here:

```
axis_count = 0
```

---

Methods inherited from [AbstractAxis](#):

**`__repr__ = __str__(self)`**

**`__str__(self)`**

**`asComponentTime(self, calendar=None)`**

Array version of cdtime tocomp. Returns a list of component times.

**`asDTGTime(self, calendar=None)`**

Array version of cdtime tocomp. Returns a list of component times.

**`asRelativeTime(self)`**

Array version of cdtime torel. Returns a list of component times.

**`assignValue(self, data)`**

**`clone(self, copyData=1)`**

clone (self, copyData=1)

Return a copy of self as a transient axis.

If copyData is 1, make a separate copy of the data.

**`designateCircular(self, modulo, persistent=0)`**

**`designateLatitude(self, persistent=0)`**

# Designate axis as a latitude axis.

# If persistent is true, write metadata to the container.

**`designateLevel(self, persistent=0)`**

# Designate axis as a vertical level axis

# If persistent is true, write metadata to the container.

**`designateLongitude(self, persistent=0, modulo=360.0)`**

# Designate axis as a longitude axis

# If persistent is true, write metadata to the container.

# If modulo is defined, set as circular

**`designateTime(self, persistent=0, calendar=4369)`**

# Designate axis as a time axis, and optionally set the calendar.

# If persistent is true, write metadata to the container.

**`genGenericBounds(self, width=1.0)`**

# Generate bounds from midpoints. width is the width of the zones.

**`getCalendar(self)`**

# Return the cdtime calendar: GregorianCalendar, NoLeapCalendar,

# or None. If the axis does not have a calendar attribute, return None.

# calendar.

**`getModulo(self)`**

```

getModuloCycle(self)

getValue(self)
    # Return the entire array

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if
    requested.

isCircular(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude
    # or latitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

isVirtual(self)
    Return true iff coordinate values are implicitly defined.

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='cn', cycle=None)
    Map coordinate interval to index interval. interval has one of
    three forms:
        (x,y)
        (x,y,indicator): indicator overrides keyword argument
        (x,y,indicator,cycle): indicator, cycle override keyword arguments
        None: indicates the full interval

    where x and y are the endpoints in coordinate space. indicator is a
    two-character string, where the first character is 'c' if the left-hand
    point is closed on the left, 'o' if open, and the second character has
    the same meaning for the right-hand point. Set cycle to a nonzero value
    to force wraparound.

```

Returns the corresponding index interval  $(i, j)$ , where  $i < j$ , in the half-open index interval  $[i, j)$ , or None if the intersection is empty.

For an axis which is circular (`self.topology == 'circular'`), is interpreted as follows (where  $N = \text{len}(\text{self})$ ):

(1) if  $j \leq N$ , the interval does not wrap around the axis endpoint.

(2) if  $j > N$ , the interval wraps around, and is equivalent to the two consecutive intervals  $[i, N]$ ,  $[0, j-N]$ .

For example, if the vector is  $[0, 2, 4, \dots, 358]$  of length 180, and the coordinate interval is  $[-5, 5)$ , the return index interval is  $[178, 183)$ . This is equivalent to the two intervals  $[178, 180)$  and  $[0, 2)$ .

Note: if the interval is interior to the axis, but does not span the entire axis element, a singleton  $(i, i+1)$  indicating an adjacent index interval.

#### **`mapIntervalExt(self, interval, indicator='ccn', cycle=None, epsilon=None)`**

Like `mapInterval`, but returns  $(i, j, k)$  where  $k$  is stride, and  $(i, j)$  is not restricted to one cycle.

#### **`rank(self)`**

#### **`setCalendar(self, calendar, persistent=1)`**

# Set the calendar

#### **`subAxis = subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

#### **`subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

#### **`toRelativeTime(self, units, calendar=None)`**

Convert time axis values to another unit

#### **`validateBounds(self, bounds)`**

# Check that a boundary array is valid, raise exception if not

---

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

#### **`dump(self, path=None, format=1)`**

dump(`self, path=None, format=1`)

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

#### **`matchPattern(self, pattern, attribute, tag)`**

# Match a pattern in a string-valued attribute. If attribute

# search all string attributes. If tag is not None, it must match.

```

matchone(self, pattern, atname)
    # Return true iff the attribute with name atname is a string
    # attribute which matches the compiled regular expression pattern
    # if atname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is not a string.

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attribute is None,
    # search all string attributes. If tag is not None, it must match the tag.

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a string
    # if the predicate is true and either tag is None or matches the tag.
    # If the predicate returns false, return an empty list.

searchone(self, pattern, atname)
    Return true iff the attribute with name atname is a string
    attribute which contains the compiled regular expression pattern
    if atname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is not a string.

```

---

Methods inherited from [cdms.internalattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)
__getattr__(self, name)
__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)
    Return the 'set' property handler for name that self uses.
    Returns None if no handler.

```

```
set_property(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)
    Set attribute handlers for name to methods actg, acts, actd
    None means no change for that action.
    nowrite = 1 prevents setting this attribute.
        nowrite defaults to 0.
    nodelete = 1 prevents deleting this attribute.
        nodelete defaults to 1 unless actd given.
    if nowrite and nodelete is None: nodelete = 1
```

## class **TransientVirtualAxis**(TransientAxis)

An axis with no explicit representation of data values.  
It appears to be a float vector with values [0.0, 1.0, ..., float(a)

Method resolution order:

TransientVirtualAxis  
TransientAxis  
AbstractAxis  
cdms.cdmsobj.CdmsObj  
cdms.internattr.InternalAttributesClass  
PropertiedClasses.Properties.PropertiedClass

---

Methods defined here:

**getitem**\_\_(self, key)

**getslice**\_\_(self, low, high)

**init**\_\_(self, axisname, axislen)

**len**\_\_(self)

**repr**\_\_ = **str**\_\_(self)

**str**\_\_(self)

**clone**(self, copyData=1)

clone (self, copyData=1)

Return a copy of self as a transient virtual axis.  
If copyData is 1, make a separate copy of the data.

**getData**(self)

**isCircular**(self)

**isVirtual**(self)

Return true iff coordinate values are implicitly defined.

**setBounds**(self, bounds)

No boundaries on virtual axes

---

Methods inherited from [TransientAxis](#):

`setitem(self, index, value)`  
`selslice(self, low, high, value)`  
`getBounds(self)`  
`getExplicitBounds(self)`  
`isLinear(self)`  
`typecode(self)`

---

Data and other attributes inherited from [TransientAxis](#):

`axis_count = 0`

---

Methods inherited from [AbstractAxis](#):

`asComponentTime(self, calendar=None)`  
Array version of cdtime tocomp. Returns a list of component times.

`asDTGTime(self, calendar=None)`  
Array version of cdtime tocomp. Returns a list of component times.

`asRelativeTime(self)`  
Array version of cdtime torel. Returns a list of component times.

`assignValue(self, data)`

`designateCircular(self, modulo, persistent=0)`

`designateLatitude(self, persistent=0)`  
# Designate axis as a latitude axis.  
# If persistent is true, write metadata to the container.

`designateLevel(self, persistent=0)`  
# Designate axis as a vertical level axis  
# If persistent is true, write metadata to the container.

`designateLongitude(self, persistent=0, modulo=360.0)`  
# Designate axis as a longitude axis  
# If persistent is true, write metadata to the container.  
# If modulo is defined, set as circular

`designateTime(self, persistent=0, calendar=4369)`  
# Designate axis as a time axis, and optionally set the calendar.  
# If persistent is true, write metadata to the container.

```

genGenericBounds(self, width=1.0)
    # Generate bounds from midpoints. width is the width of the z

getCalendar(self)
    # Return the cdtime calendar: GregorianCalendar, NoLeapCalendar
    # or None. If the axis does not have a calendar attribute, return
    # calendar.

getModulo(self)

getModuloCycle(self)

getValue(self)
    # Return the entire array

info(self, flag=None, device=None)
    Write info about axis; include dimension values and weights if applicable

isCircularAxis(self)
    # Return true iff the axis wraps around
    # An axis is defined as circular if:
    # (1) self.topology=='circular', or
    # (2) self.topology is undefined, and the axis is a longitude axis

isLatitude(self)
    # Return true iff the axis is a latitude axis

isLevel(self)
    # Return true iff the axis is a level axis

isLongitude(self)
    # Return true iff the axis is a longitude axis

isTime(self)
    # Return true iff the axis is a time axis

listall(self, all=None)
    Get list of info about this axis.

mapInterval(self, interval, indicator='ccn', cycle=None)
    Map coordinate interval to index interval. interval has one character
    (x,y)
    (x,y,indicator): indicator overrides keyword argument
    (x,y,indicator,cycle): indicator, cycle override keyword arguments
    None: indicates the full interval

    where x and y are the endpoints in coordinate space. indicator is a
    two-character string, where the first character is 'c' if the left-hand
    is closed on the left, 'o' if open, and the second character has the
    same meaning for the right-hand point. Set cycle to a nonzero value
    to force wraparound.

```

Returns the corresponding index interval  $(i, j)$ , where  $i < j$ , in the half-open index interval  $[i, j)$ , or None if the intersection is empty.

For an axis which is circular (`self.topology == 'circular'`), is interpreted as follows (where  $N = \text{len}(\text{self})$ ):

- (1) if  $j \leq N$ , the interval does not wrap around the axis endpoint.
- (2) if  $j > N$ , the interval wraps around, and is equivalent to the two consecutive intervals  $[i, N]$ ,  $[0, j-N)$ .

For example, if the vector is  $[0, 2, 4, \dots, 358]$  of length 180, and the coordinate interval is  $[-5, 5)$ , the return index interval is  $[178, 183)$ . This is equivalent to the two intervals  $[178, 180)$  and  $[0, 2)$ .

Note: if the interval is interior to the axis, but does not span the entire axis element, a singleton  $(i, i+1)$  indicating an adjacent index interval.

**`mapIntervalExt(self, interval, indicator='ccn', cycle=None, epsilon=None)`**

Like `mapInterval`, but returns  $(i, j, k)$  where  $k$  is stride, and  $(i, j)$  is not restricted to one cycle.

**`rank(self)`**

**`setCalendar(self, calendar, persistent=1)`**

# Set the calendar

**`subAxis = subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

**`subaxis(self, i, j, k=1)`**

Create a transient axis for the index slice  $[i:j:k]$

The stride  $k$  can be positive or negative. Wraparound is supported for longitude dimensions or those with a modulus attribute.

**`toRelativeTime(self, units, calendar=None)`**

Convert time axis values to another unit

**`validateBounds(self, bounds)`**

# Check that a boundary array is valid, raise exception if not

---

Methods inherited from [cdms.cdmsobj.CdmsObj](#):

**`dump(self, path=None, format=1)`**

`dump(self, path=None, format=1)`

Dump an XML representation of this object to a file.

'path' is the result file name, None for standard output.

'format'==1 iff the file is formatted with newlines for readability.

**`matchPattern(self, pattern, attribute, tag)`**

```

# Match a pattern in a string-valued attribute. If attribute
# search all string attributes. If tag is not None, it must m

matchone(self, pattern, atname)
    # Return true iff the attribute with name atname is a string
    # attribute which matches the compiled regular expression pat
    # if atname is None and pattern matches at least one string
    # attribute. Return false if the attribute is not found or is n

searchPattern(self, pattern, attribute, tag)
    # Search for a pattern in a string-valued attribute. If attri
    # search all string attributes. If tag is not None, it must m

searchPredicate(self, predicate, tag)
    # Apply a truth-valued predicate. Return a list containing a
    # if the predicate is true and either tag is None or matches
    # If the predicate returns false, return an empty list

searchone(self, pattern, atname)
    Return true iff the attribute with name atname is a string
    attribute which contains the compiled regular expression patt
    if atname is None and pattern matches at least one string
    attribute. Return false if the attribute is not found or is n
    a string.

```

---

Methods inherited from [cdms.internattr.InternalAttributesClass](#):

```

is_internal_attribute(self, name)
    is internal attribute(name) is true if name is internal.

replace_external_attributes(self, newAttributes)
    replace external attributes(newAttributes)
    Replace the external attributes with dictionary newAttributes

```

---

Methods inherited from [PropertiedClasses.Properties.PropertiedClass](#):

```

__delattr__(self, name)

__getattr__(self, name)

__setattr__(self, name, value)

get_property_d(self, name)
    Return the 'del' property handler for name that self uses.
    Returns None if no handler.

get_property_g(self, name)
    Return the 'get' property handler for name that self uses.
    Returns None if no handler.

get_property_s(self, name)

```

Return the 'set' property handler for name that self uses.  
Returns None if no handler.

***set\_property*(self, name, actg=None, acts=None, actd=None, nowrite=None, nodelete=None)**

Set attribute handlers for name to methods actg, acts, actd  
None means no change for that action.  
nowrite = 1 prevents setting this attribute.  
nowrite defaults to 0.  
nodelete = 1 prevents deleting this attribute.  
nodelete defaults to 1 unless actd given.  
if nowrite and nodelete is None: nodelete = 1

## Functions

***allclose(ax1, ax2, rtol=1.000000000000001e-05, atol=1e-08)***

True if all elements of axes ax1 and ax2 are close,  
in the sense of MA.allclose.

***axisMatchAxis(axes, specifications=None, omit=None, order=None)***

Given a list of axes and a specification or list of  
specifications, and a specification or list of specifications  
of those axes to omit, return a list of  
those axes in the list that match the specification but  
do not include in the list any axes that matches an omit  
specification.

If specifications is None, include all axes less the omitted ones.

Individual specifications must be integer indices into axes or  
matching criteria as detailed in axisMatches.

Axes are returned in the order they occur in the axes argument until  
order is given.

order can be a string containing the symbols t,x,y,z, or -.

If a - is given, any elements of the result not chosen otherwise are  
filled in from left to right with remaining candidates.

***axisMatchIndex(axes, specifications=None, omit=None, order=None)***

Given a list of axes and a specification or list of  
specifications, and a specification or list of specifications  
of those axes to omit, return a list of the indices of  
those axes in the list that match the specification but  
do not include in the list any axes that matches an omit  
specification.

If specifications is None, include all axes less the omitted ones.

Individual specifications must be integer indices into axes or  
matching criteria as detailed in axisMatches.

The indices of axes are returned in the order the axes occur in the axes argument, unless order is given.

order can be a string containing the symbols t,x,y,z, or -. If a - is given, any elements of the result not chosen otherwise are filled in from left to right with remaining candidates.

#### ***axisMatches*(axis, specification)**

Return 1 or 0 depending on whether axis matches the specification. Specification must be one of:

1. a string representing an axis id or one of the keywords time, latitude or lat, longitude or lon, or lev or level.

axis may be surrounded with parentheses or spaces.

We first attempt to match the axis id and the specification.

Keywords try to match using isTime, isLatitude, etc.

Comparisons to keywords and axis ids is case-insensitive.

2. a function that takes an axis as an argument and returns a value if the value returned is true, the axis matches.

3. an axis object; will match if it is the same object as axis.

#### ***concatenate*(axes, id=None, attributes=None)**

Concatenate the axes, return a transient axis.

#### ***createAxis*(data, bounds=None, id=None, copy=0)**

# Create a transient axis

#### ***createEqualAreaAxis*(nlat)**

# Generate an equal-area latitude axis, north-to-south

#### ***createGaussianAxis*(nlat)**

# Generate a Gaussian latitude axis, north-to-south

#### ***createUniformLatitudeAxis*(startLat, nlat, deltaLat)**

# Generate a uniform latitude axis

#### ***createUniformLongitudeAxis*(startLon, nlon, deltaLon)**

# Generate a uniform longitude axis

#### ***getAutoBounds*()**

#### ***isOverlapVector*(vec1, vec2, atol=1e-08)**

Returns (isoverlap, index) where:

isoverlap is true iff a leading portion of vec1 is a subset of vec2. index is the index such that vec1[0]<=vec2[index]. If index==len(vec2), then vec1[0]>vec2[len(vec2)-1]

```

isSubsetVector(vec1, vec2, tol)
    # Return true if vector vec1 is a subset of vec2, within tolerance tol
    # Return second arg of index, if it is a subset

lookupArray(ar, value)
    Lookup value in array ar. Return index such that:
    (a) ar is monotonically increasing:
        value <= ar[index], index==0..len(ar)-1
        value > ar[index], index==len(ar)
    (b) ar is monotonically decreasing:
        value >= ar[index], index==0..len(ar)-1
        value < ar[index], index==len(ar)

mapLinearExt(axis, bounds, interval, indicator='ccn', epsilon=None, stride=1, wrapped=0)
    Map coordinate interval to index interval, without wraparound. interval has the form (x,y) where x and y are the endpoints in coordinate space. indicator is a three-character string, where the first character is 'c' if the interval is closed on the left, 'o' if open, and the second character has the same meaning for the right-hand point. The third character indicates how the intersection of the interval and axis is treated:

    'n' - the node is in the interval
    'b' - the interval intersects the cell bounds
    's' - the cell bounds are a subset of the interval
    'e' - same as 'n', plus an extra node on either side.

    Returns the corresponding index interval (i,j), where i<j, indicating the half-open index interval [i,j), or None if the intersection is empty.

mapLinearIntersection(xind, yind, iind, aMinusEps, aPlusEps, bPlusEps, bMinusEps, boundLeft, nodeSubI, boundRight)
    Return true iff the coordinate interval (a,b) intersects the node nodeSubI or cell bounds [boundLeft,boundRight], where the interval (a,b) is defined by:

    xind = 'c' if (a,b) is closed on the left, 'o' if open,
    yind same for right endpoint
    aMinusEps,aPlusEps = a +/- epsilon
    bPlusEps,bMinusEps = b +/- epsilon

    and the intersection option iind = 'n','b','e','s' specifies whether the intersection is with respect to the node value nodeSubI ('n' or 'e') or the cell bounds [boundLeft,boundRight]. See mapLinearExt.

setAutoBounds(mode)
    # Set autobounds mode to 'on' or 'off'. If on, getBounds will auto-
    # generate boundary information for an axis or grid, if not explicit
    # If 'off', and no boundary data is explicitly defined, the bounds
    # be generated; getBounds will return None for the boundaries.

```

***take***(ax, indices)

Take values indicated by indices list, return a transient axis.

## Data

```
CdtimeTypes = (<type 'comptime'>, <type 'reltime'>)
FileWasClosed = 'File was closed for object: '
InvalidBoundsArray = 'Invalid boundary array: '
InvalidCalendar = 'Invalid calendar: '
InvalidNCycles = 'Invalid number of cycles requested for wrapped dimension: '
MethodNotImplemented = 'Method not yet implemented'
ReadOnlyAxis = 'Axis is read-only: '
calendarToTag = {17: '360_day', 4113: 'noleap', 4369: 'proleptic_gregorian', 69905: 'julian',
135441: 'gregorian'}
latitude_aliases = []
level_aliases = ['plev']
longitude_aliases = []
std_axis_attributes = ['name', 'units', 'length', 'values', 'bounds']
tagToCalendar = {'360': 17, '360_day': 17, '365_day': 4113, 'gregorian': 135441, 'julian': 69905,
'noleap': 4113, 'proleptic_gregorian': 4369, 'standard': 4369}
time_aliases = []
unspecified = 'No value specified.'
```